Programación Estructurada Matemáticamente

Mauro Jaskelioff

Fundamentos y aplicaciones de la lógica y la programación - CIFASIS-CONICET
Depto. de Computación - FCEIA - Universidad Nacional de Rosario

jaskelioff@cifasis-conicet.gov.ar

Resumen

Los lenguajes de programación modernos poseen características que aumentan considerablemente su poder expresivo. Este poder expresivo permite la internalización de las estructuras matemáticas que sustentan la semántica de los lenguajes. Dichas estructuras proveen abstracciones que facilitan distintos aspectos del desarrollo de software, como ser modularidad, optimización, seguridad y verificación. Nos proponemos investigar las propiedades y aplicaciones de estructuras tales como mónadas, arrows, y functores aplicativos.

Palabras Clave: Lenguajes – Semántica – Modelos Matemáticos – Modularidad

1. Contexto

Esta línea de investigación se desarrolla en el CIFASIS, instituto de investigación dependiente del CONICET. Mantenemos una colaboración con la Universidad de Nottingham y la Universidad de Strathclyde del Reino Unido, y con la Universidad de Tecnología Chalmers, de Suecia. Cuenta con financiamiento de la Agencia de Promoción Científica y Tecnológica.

2. Introducción

La dificultad y el alto costo del desarrollo de grandes sistemas de software ha sido bien documentada (por ejemplo, [1] y [12]). Por lo tanto, es fundamental para la industria en general, y para la industria del software en particular, el desarrollo de técnicas y métodos que faciliten el diseño, implementación, verificación y testeo de los sistemas de software.

La teoría de los lenguajes de programación facilita el desarrollo de software mediante el desarrollo de mejores lenguajes de programación. En los lenguajes modernos el significado de un programa existe más allá de su implementación en un compilador o intérprete. Esto permite **razonar efectivamente** acerca de los programas y, por ejemplo, optimizar un programa mediante su transformación a otro programa más eficiente pero con igual comportamiento, sin necesidad de razonar acerca de su compilación a un lenguaje de bajo nivel.

La semántica *denotacional* de un lenguaje de programación es una herramienta muy poderosa para poder descubrir sutilezas en la interacción entre distintas características del lenguaje. Por otro lado, las mismas estructuras que organizan la semántica sirven también para **organizar la implementación** de los intérpretes y compiladores. El uso de **mónadas** para estructurar semánticas denotacionales [26, 27] y como método de programación [31, 32] ha tenido un fuerte impacto ya que, por primera vez, permitió modelar con una única abstracción efectos computacionales tan diversos como ser excepciones, continuaciones, entrada/salida, almacenamiento en memoria, paralelismo y concurrencia. En particular, el lenguaje Haskell posee una notación especial para escribir programas monádicos y toda su entrada/salida se realiza mediante una mónada [28]. Sin embargo **la utiliza-**

ción de mónadas no se limita a Haskell y es también utilizada en otros lenguajes (generalmente declarativos) tales como ML y Lisp.

Luego de la popularización de las mónadas, se empezaron a utilizar otras estructuras, entre las cuales se destacan las **arrows** [16] y los **functores aplicativos** [25]. Las arrows fueron introducidas para tratar casos donde el uso de mónadas no era posible o donde estas últimas estructuras pueden lograr una mayor eficiencia. En particular, las arrows han sido utilizados para la programación funcional reactiva [8] (por ej. para interfaces gráficas o robótica) y para el diseño de lenguajes orientados al flujo de datos (por ej. circuitos [24]) y para obtener combinadores de parsers eficientes [16]. Los functores aplicativos son una generalización de las mónadas. Permiten trabajar en un estilo aplicativo que es más natural para los programadores funcionales.

El estudio de estas estructuras semánticas no es sólo de la incumbencia de semanticistas y desarrolladores de procesadores de lenguajes de programación. En algún sentido, todos los programadores escriben procesadores de lenguajes. Según Reynolds, "cualquier proceso que acepte información de usuarios humanos es un procesador de lenguajes" [29]. Esta visión de la programación es hoy en día todavía más relevante con la creciente popularidad de los lenguajes de dominio específico (DSL), en donde el programador debe crear un lenguaje orientado a una aplicación en particular. En este enfoque para el desarrollo de software, el lenguaje provee una sintaxis y semántica adecuada para el dominio, por lo que es accesible a no expertos y además permite la incorporación de optimizaciones específicas del dominio.

El desarrollo de DSLs puede tornarse tedioso ya que probablemente haya que implementar características comunes a la mayoría de los lenguajes, como ser operaciones aritméticas u operaciones sobre booleanos. Por este motivo, se han popularizado los lenguajes **embebidos** de dominio específico (EDSL). En este caso, el DSL se embebe en algún lenguaje de programación de propósito general. Consecuentemente el implementador del EDSL no tiene que lidiar con parsers, librerías, manejo de errores, compilación, etc. En los DSL, y particularmente en los EDSL, estructurar las implementaciones con mónadas, arrows o functores aplicativos tiene enormes beneficios a la hora de estructurar, modularizar, optimizar, verificar y mantener los programas. Por lo tanto, tanto los aspectos teóricos como las técnicas de implementación de las estructuras semánticas son potencialmente relevantes para cualquier programador.

3. Líneas de Investigación y desarrollo

- Teoría:
  - Modularidad en estructuras semánticas. Estudiamos la forma de combinar e incrementar funcionalidad en mónadas, arrows y functores aplicativos. De esta manera los programas que utilizan estas estructuras se pueden modularizar y desarrollar incrementalmente. En particular, estas combinaciones permitirían la construcción modular de DSLs.
  - Fusión de programas. Los programas escritos en forma composicional pueden ser ineficientes, ya que en una composición $(f \circ g)$, la función g puede generar una estructura compleja, sólo para ser inmediatamente consumida por la función f. La fusión de programas permite la manipulación de programas mediante reglas algebraicas para, a partir de una composición de funciones, obtener una función monolítica equivalente que no genere la estructura intermedia.
- Aplicaciones:
  - Diseño e implementación de un lenguaje para la programación de agentes que permita incorporar el modelo graduado de BDI, estructurando la implementación con arrows. Las arrows han demostrado ser una estructura apropiada para la modelización de programas funcionales reactivos. Al implementar una arquitectura de agentes en un lenguaje de alto nivel, pretendemos poder experimentar de una forma modular, modifi-

caciones en sus componentes. Se quiere analizar también cuales de las extensiones introducidas por miembros del grupo en la arquitectura BDI graduada puede incorporarse en los lenguajes de programación de agentes.

- Lenguajes para proveer seguridad en software. Desarrollo de un EDSL (lenguaje de dominio específico embebido) para asegurar la confidencialidad de programas. Mediante el uso de este EDSL, uno puede garantizar que información privada no se filtre hacia canales públicos.

4. Resultados y Objetivos

Motivado por el estudio de la modularidad en mónadas se estudiaron los transformadores de monoides en una categoría monoidal. Trabajando a este nivel de abstracción se obtuvieron varios teoremas de promoción de operaciones del monoide existente al monoide transformado [22, 19]. Es bien sabido que las mónadas son monoides en la categoría monoidal de endofunctores, por lo que estos resultados son directamente aplicables al estudio de transformadores de mónadas. En [20], la teoría se desarrolla en el contexto de mónadas expresables en el sistema Fω.

Por otro lado las arrows son monoides en la categoría monoidal de profunctores (también llamados distribuidores) [15], por lo que la teoría sería aplicable también al análisis de modularidad en arrows.

Este desarrollo teórico ya ha sido llevada a la práctica para el caso de mónadas con la biblioteca Monatron [18], que desarrolla un DSL embebido para la construcción de computaciones monádicas modulares. Esta biblioteca, presenta varias mejoras con respecto a las bibliotecas de transformadores de mónadas existentes en términos de extensibilidad, regularidad de su semántica y expresibilidad.

Se ha desarrollado una técnica general para la optimización de semánticas por algebra inicial [17]. Este da un marco para estudiar optimizaciones mas específicas en el contexto de DSLs. En particular una de las optimizaciones estudiadas mejora la eficiencia de la operación *bind* (correspondiente a la extensión de Kleisli de la mónada). El caso de esta optimización aplicada al caso particular de la mónada libre fue publicada en [30]. La técnica presentada en [17] permitió probar formalmente la correción de la optimización y generalizar el resultado.

También se ha analizado la implementación y modularidad de semánticas operacionales de manera tal que se obtiene una formulación general que captura la estructura fundamental de los sistemas de transición con un buen comportamiento [21]. Esta semántica modular se puede entender también como una disciplina para razonar sobre lenguajes definidos por semántica de álgebra inicial sobre un álgebra con portador definido coinductivamente.

La deforestación de programas consiste en eliminar la estructura intermedia de una composición $f \circ g$, cuando f es un "fold" (consume el tipo de datos intermedio uniformemente) y g es un "build" (genera el tipo de datos uniformemente) [13]. Esta técnica ha sido extendida para el caso de programas monádicos [9, 10] por otros autores. En un trabajo reciente [7] hemos propuesto una ley de deforestación para computaciones estructuradas usando functores aplicativos. El caso de deforestación para arrows todavía no ha sido desarrollado.

Una de las aplicaciones de la programación estructurada matemáticamente que pensamos desarrollar es un lenguaje para la programación de agentes. Casali et al. han planteado en [5, 3, 4, 2, 6] un modelo BDI graduado (g-BDI) considerando que esta arquitectura permite diseñar e implementar agentes capaces de tener mejor performance en entornos dinámicos e inciertos. Esta arquitectura se basa en los sistemas multicontextos [11] y es una extensión al modelo BDI, con una semántica más rica, permitiendo elegir distintas lógicas y medidas de incertidumbre para representar el comportamiento de las actitudes mentales [14]. El contar con una especificación modular de alto nivel como la que se plantea desarrollar, permitiría implementar el modelo g-BDI básico y alguna de sus variantes, facilitando la experimentación y compararación entre las distintas arquitecturas posibles, permitiendo la selección de la más adecuada para definir un agente concreto. También se podrá probar y experimentar con las propuestas de revisión de los distintos contextos, en las cuales se está trabajan-

do. Por otro lado, es de suponer que la implementación pondrá a prueba diferentes aspectos teóricos, con lo que la aplicación realimentará a la teoría, proveyéndole nuevos desafíos.

Otra de las aplicaciones que estamos desarrollando concierne el desarrollo de software seguro. La necesidad de desarrollar software que permita asegurar la confidencialidad de datos ha aumentado de la mano del enorme crecimiento de servicios en línea. Hemos desarrollado un EDSL para asegurar la confidencialidad de programas escritos en Haskell [23]. Esta implementación permite mostrar la viabilidad de la idea pero debe ser extendida para implementar todas las operaciones primitivas de entrada/salida de Haskell. Por otro lado, para poder razonar más efectivamente sobre la implementación y verificar sus propiedades es necesario el desarrollo de una máquina virtual que de semántica en forma pura a las distintas operaciones de entrada/salida.

5. Formación de Recursos Humanos

El equipo se compone actualmente del Dr. Mauro Jaskelioff que dirige esta línea de investigación, el Lic. Dante Zanarini está haciendo un doctorado utilizando mónadas para implementar una semántica modificada de I/O que asegura la confidencialidad de la información privada. El Lic. Germán Andrés Delbianco terminó su tesina en Diciembre 2010 sobre recursión estructurada con functores aplicativos. La Dra. Ana Casali y el doctorando Pablo Pilotti participan con lo relevante a modelos BDI de agentes, arquitecturas g-BDI y lenguajes para la programación de agentes.

Referencias

[1] F.P. Brooks. No silver bullet. *IEEE Computer*, 20(4):10–19, 1987.

[2] Ana Casali. *On Intentional and Social Agents with graded Attitudes*. Monografíes de L'Institut D'Investigacó en Intel.ligència Artificial IIIA-CSIC, 2009.

[3] Ana Casali, Lluís Godo, and Carles Sierra. Graded bdi models for agent architectures. In João Alexandre Leite and Paolo Torroni, editors, *CLIMA V*, volume 3487 of *Lecture Notes in Computer Science*, pages 126–143. Springer, 2004.

[4] Ana Casali, Lluís Godo, and Carles Sierra. Multi-context specification for graded bdi agents. In *Proceedings of the Doctoral Consortium - Fifth International Conference on Modeling and Using Context*, Paris, France, 2005.

[5] Ana Casali, Lluís Godo, and Carles Sierra. Modelos bdi graduados para arquitecturas de agentes. *Revista Iberoamericana de Inteligencia Artificial.*, 9(26):67–75, 2006.

[6] Ana Casali, Lluís Godo, and Carles Sierra. g-bdi: A graded intensional agent model for practical reasoning. volume 5861, pages 5–20, Awaji Island, Japan, 11 2009. Springer, Springer.

[7] Germán Andrés Delbianco, Mauro Jaskelioff, and Alberto Pardo. Applicative shortcut fusion. In *Trends in Functional Programming (TFP'11)*, Madrid, Spain, 2011.

[8] C. Elliott and P. Hudak. Functional reactive animation. In *Proceedings of the second ACM SIGPLAN international conference on Functional programming*, pages 263–273. ACM New York, NY, USA, 1997.

[9] N. Ghani, T. Uustalu, and V. Vene. Build, augment and destroy, universally. In *Programming languages and systems: Second Asian Symposium, APLAS 2004, Taipei, Taiwan, November 4-6, 2004: proceedings*, page 327. Springer-Verlag New York Inc, 2004.

[10] Neil Ghani, Patricia Johann, Tarmo Uustalu, and Varmo Vene. Monadic augment and generalised short cut fusion. In Olivier Danvy and Benjamin C. Pierce, editors, *ICFP*, pages 294–305. ACM, 2005.

[11] C. Ghidini and F. Giunchiglia. Local models semantics, or contextual reasoning= locality+

compatibility. *Artificial intelligence*, 127(2):221–259, 2001.

[12] W.W. Gibbs. Software's chronic crisis. *Scientific American*, 271(3):72–81, 1994.

[13] A. Gill, J. Launchbury, and S.L.P. Jones. A short cut to deforestation. In *Proceedings of the conference on Functional programming languages and computer architecture*, pages 223–232. ACM New York, NY, USA, 1993.

[14] L. Godo, F. Esteva, and P. Hajek. A fuzzy modal logic for belief functions. *Fundamenta Informaticae archive*, 57(2-4):127–146, 2003.

[15] Chris Heunen and Bart Jacobs. Arrows, like monads, are monoids. *Electronic Notes in Theoretical Computer Science*, 158:219–236, 2006.

[16] John Hughes. Generalising monads to arrows. *Science of Computer Programming*, 37(1-3):67–111, 5 2000.

[17] Graham Hutton, Mauro Jaskelioff, and Andy Gill. Factorising folds for faster functions. Accepted for publication in the Journal of Functional Programming special issue on Generic Programming, 10 2009.

[18] Mauro Jaskelioff. Monatron: an extensible monad transformer library. In *Implementation and Application of Functional Languages*, 2008. Accepted for publication.

[19] Mauro Jaskelioff. *Lifting of Operations in Modular Monadic Semantics*. PhD thesis, University of Nottingham, 2009.

[20] Mauro Jaskelioff. Modular monad transformers. In Giuseppe Castagna, editor, *European Symposium on Programming*, volume 5502 of *Lecture Notes in Computer Science*, pages 64–79. Springer, 2009.

[21] Mauro Jaskelioff, Neil Ghani, and Graham Hutton. Modularity and implementation of mathematical operational semantics. In *Proceedings of the Workshop on Mathematically Structured Functional Programming*, Reykjavik, Iceland, 7 2008.

[22] Mauro Jaskelioff and Eugenio Moggi. Monad transformers as monoid transformers. *Theoretical Computer Science*, 2010. Accepted for publication.

[23] Mauro Jaskelioff and Alejandro Russo. Secure multi-execution in haskell. In *Ershov Informatics Conference*, Novosibirsk, Akademgorodok, Russia, 2011.

[24] J. Launchbury, J.R. Lewis, and B. Cook. On embedding a microarchitectural design language within Haskell. In *Proceedings of the fourth ACM SIGPLAN international conference on Functional programming*, page 69. ACM, 1999.

[25] Conor McBride and Ross Paterson. Applicative programming with effects. *Journal of Functional Programming*, 18(01):1–13, 2007.

[26] Eugenio Moggi. Computational lambda-calculus and monads. In *LICS*, pages 14–23. IEEE Computer Society, 1989.

[27] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

[28] Simon L. Peyton Jones and Philip Wadler. Imperative functional programming. In *POPL*, pages 71–84, 1993.

[29] John C. Reynolds. *Theories of Programming Languages*. Cambridge University Press, 1998.

[30] Janis Voigtländer. Asymptotic improvement of computations over free monads. In *MPC*, pages 388–403, 2008.

[31] Philip Wadler. Comprehending monads. *Mathematical Structures in Computer Science*, 2(4):461–493, 1992.

[32] Philip Wadler. The essence of functional programming. In *POPL*, pages 1–14, 1992.